Computational Statistics
Lab 2

Group 03
Mohsen Pirmoradiyan, Ahmed Alhasan, Yash Pawar

07 Feb 2020

Question 1: Optimizing a model parameter

The file mortality rate.csv contains information about mortality rates of the fruit flies during a certain period.

1. Import this file to R and add one more variable LMR to the data which is the natural logarithm of Rate. Afterwards, divide the data into training and test sets by using the following code:

n=dim(data)[1]
set.seed(123456)
id=sample(1:n, floor(n * 0.5))
train=data[id,]
test=data[-id,]

```
mortality <- read.csv2("../Data/mortality_rate.csv")
mortality$LMR <- log(mortality$Rate)

n <- dim(mortality)[1]
RNGversion(min(as.character(getRversion()),"3.6.2")) ## with your R-version
set.seed(12345, kind = "Mersenne-Twister", normal.kind = "Inversion")
id <- sample(1:n, floor(n * 0.5))
train <- mortality[id,]
test  <- mortality[-id,]
```

2. Write your own function myMSE() that for given parameters λ and list pars containing vectors X, Y, Xtest, Ytest fits a LOESS model with response Y and predictor X using loess() function with penalty λ (parameter enp.target in loess()) and then predicts the model for Xtest. The function should compute the predictive MSE, print it and return as a result. The predictive MSE is the mean square error of the prediction on the testing data. It is defined by the following Equation (for you to implement):

$$predictive\ MSE = \frac{1}{length(test)} \sum_{ith\ element\ in\ the\ test\ set} (Ytest[i] - fYpred(X[i]))^2$$

where fYpred(X[i]) is the predicted value of Y if X is X[i]. Read on R's functions for prediction so that you do not have to implement it yourself.

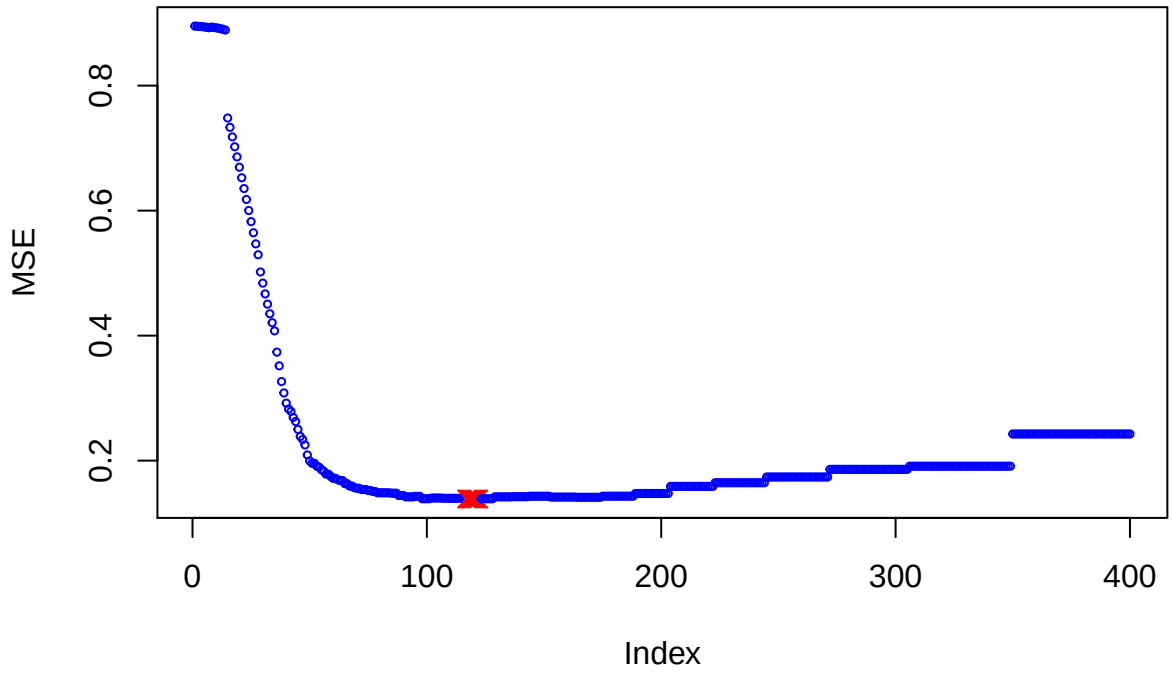
```
pars  <- list(X = train$Day,
              Y = train$LMR,
              Xtest = test$Day,
              Ytest = test$LMR)

myMSE <- function(lambda, pars){
    model <- loess(pars$Y ~ pars$X, enp.target = lambda)
    Ypred <- predict(model, newdata = pars$Xtest)
    MSE   <- sum((pars$Ytest - Ypred)^2) / dim(test)[1]
    k     <<- k + 1
    return(MSE)
}
```

3. Use a simple approach: use function myMSE(), training and test sets with response LMR and predictor Day and the following λ values to estimate the predictive MSE values: $\lambda = 0.1, 0.2, \ldots, 40$

```
lambda <- seq(0.1, 40, 0.1)
MSE <- numeric()
k   <- 0
for(i in 1:length(lambda)){
  MSE[i] <- myMSE(lambda[i],pars)
}
```

4. Create a plot of the MSE values versus λ and comment on which λ value is optimal. How many evaluations of myMSE() were required (read ?optimize) to find this value?



```
## $`optimal MSE`
## [1] 0.1386422
##
## $`optimal lambda value`
## [1] 11.7
##
## $`optimal lambda location in the sequence`
## [1] 117
##
## $`Number of Iterations`
## [1] 400
```

5. Use optimize() function for the same purpose, specify range for search [0.1, 40] and the accuracy 0.01. Have the function managed to find the optimal MSE value? How many myMSE() function evaluations were required? Compare to step 4.

```
## $`optimal MSE`
## [1] 0.138957
##
## $`optimal lambda location in the sequence`
## [1] 12.68492
##
## $`Number of Iterations`
## [1] 13
```

- In step 4 we had to count all the values of MSE to find the minimum MSE (what is called exhaustive search method) so the total number of iterations is 400, while optimize() function uses a combination of Golden Section Search and Successive Parabolic Interpolation which converge faster (13 iterations) to the minimum MSE (which are multiple values at the minimum).

6. Use optim() function and BFGS method with starting point $\lambda = 35$ to find the optimal λ value. How many myMSE() function evaluations were required (read ?optim)? Compare the results you obtained with the results from step 5 and make conclusions.

```
## $`optimal MSE`
## [1] 0.2425965
##
## $`Number of  Iterations`
## [1] 3
##
## $`Number of function evaluations`
## [1] 1
##
## $`Number of gradient evaluations`
## [1] 1
```

- Although optim() called myMSE() 3 times, it didn't count the number of finite-difference approximations to the gradient, that's why it shows only 1.
- In comparasion with the previous method (derivative free method), gradient methods are much faster given that the function is differentiable. However, because our variable (lambda) is discrete the BFGS algorithm stuck at nearest point where the derivative is 0, which in this case the derivation at the starting point is 0 so it return the MSE value at that point.

Question 2: Maximizing likelihood

The file data.RData contains a sample from normal distribution with some parameters μ, σ. For this question read ?optim in detail.

1. Load the data to R environment.

```
load("C:/Users/WizzCon/Desktop/Machine Learning/Workshop/6. Computational Statistics/1. Labs/Data/data.
```

2. Write down the log-likelihood function for 100 observations and derive maximum likelihood estimators for μ, σ analytically by setting partial derivatives to zero. Use the derived formulae to obtain parameter estimates for the loaded data.

$$L(\mu, \sigma^2|X) = \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} \exp -(\frac{(x_i - \mu)^2}{2\sigma^2}) = \frac{1}{(\sigma\sqrt{2\pi})^N} \exp -\left(\frac{1}{2\sigma^2}\sum_{i=1}^{N}(x_i - \mu)^2\right)$$

$$l(\mu, \sigma^2|X) = \log\left(\frac{1}{(\sigma\sqrt{2\pi})^N}\right) - \frac{1}{2}\sum_{i=1}^{N}\frac{(x_i - \mu)^2}{\sigma^2}$$

Taking the derivative with respect to μ and set it to 0.

Good.

$$\frac{\partial l}{\partial \mu} = 0 \quad \rightarrow \quad -\frac{1}{2\sigma^2}\sum_{i=1}^{N}(2\mu - 2x_i) = 0$$

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Taking the derivative with respect to sigma and set it to 0.

$$\frac{\partial l}{\partial \sigma} = 0 \quad \rightarrow \quad -\frac{N}{\sigma} + \sum_{i=1}^{N}(x_i - \mu)^2\sigma^{-3} = 0$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

```
N     <- length(data)
mu    <- (1/N) * sum(data)
sigma <- sqrt((1/N) * sum((data-mu)^2))
paste("Mu =", mu)
```

```
## [1] "Mu = 1.27552760103638"
```

```r
paste("Sigma =", sigma)
```

```
## [1] "Sigma = 2.00597647210603"
```

3. Optimize the minus log-likelihood function with initial parameters $\mu = 0$, $\sigma = 1$. Try both Conjugate Gradient method (described in the presentation handout) and BFGS (discussed in the lecture) algorithm with gradient specified and without. Why it is a bad idea to maximize likelihood rather than maximizing log-likelihood?

- When we have large number of observations, the value of the first part tends to be very small and consequently will be rounded to zero by machine. The same story may happened for the exponential part as the exponent tends to -infinity. So it is bad idea to use this formulation when numerical methods are to use for optimization.

4. Did the algorithms converge in all cases? What were the optimal values of parameters and how many function and gradient evaluations were required for algorithms to converge? Which settings would you recommend?

```r
like <- function(pars,data){
  N   <- length(data)
  mu  <- pars[1]
  sd  <- pars[2]
  log <- -(1/(sd * (sqrt(2*pi)))^N) * exp(-((1/2 * sd^2) * sum((data-mu)^2)))
  return(log)
}
print(like(pars = c(mu,sigma), data), digits = 22)
```

```
## [1] 0
```

```r
log_like <- function(pars,data){
  N   <- length(data)
  mu  <- pars[1]
  sd  <- pars[2]
  log <- -log(1/(sd * (sqrt(2*pi)))^N) + (1/2) * sum((data-mu)^2/sd^2)
  k   <<- k + 1
  return(log)
}
log_like(pars = c(mu,sigma), data)
```

```
## [1] 211.5069
```

```r
gr <- function(pars,data){
  N  <- length(data)
  mu <- pars[1]
  sd <- pars[2]
  mu_gr <- -1 * -(sum(mu-data) / sd^2)
  sd_gr <- -1 * ((-N/sd) + sum((data-mu)^2) / sd^3)
  gr <- c(mu_gr,sd_gr)
  t  <<- t + 1
  return(gr)
}
t <- 0
gr(pars = c(mu,sigma), data)
```

```
## [1] -3.862661e-16  0.000000e+00
```

Good table and interpretation.

	Mu	Sigma	Negative Log-likelihood
CG w/o gradient	1.275528	2.005977	211.5069
BFGS w/o gradient	1.275527	2.005977	211.5069
CG with gradient	1.275528	2.005977	211.5069
BFGS with gradient	1.275528	2.005977	211.5069

	function evaluations	gradient evaluations	function calls	gradient calls
CG w/o gradient	119	23	211	0
BFGS w/o gradient	37	15	97	0
CG with gradient	53	17	53	17
BFGS with gradient	39	15	39	15

- The algorithms converged to the true value of μ and σ in all cases because it is generated from a normal distribution where the log-likelihood function is a continuous differentiable function that has only one global maximum.
- The conjugate-gradient method is a low storage algorithm because it does not involve calculating the hessian matrix, while the BFGS algorithm does not calculate the hessian matrix it approximate it and this make the iterations of BFGS computationally more expensive, compared to the CG and the stored information in approximated hessian make the BFGS use less iterations.
- Deriving and calculating the gradient analytically is better than using finite difference approximations (in the table we can see the function calls=function evaluations when gradient function is used since no finite difference approximations were used).
- Conclusion, both CG and BFGS are good methods, however because we only have 2 variables and small data the BFGS is slightly better since the approximate hessian is small. Therefore; BFGS with gradient is the best option.

Appendix

```
##Question 1: Optimizing a model parameter
mortality <- read.csv2("../Data/mortality_rate.csv")
mortality$LMR <- log(mortality$Rate)

n <- dim(mortality)[1]
RNGversion(min(as.character(getRversion()),"3.6.2")) ## with your R-version
set.seed(12345, kind = "Mersenne-Twister", normal.kind = "Inversion")
id <- sample(1:n, floor(n * 0.5))
train <- mortality[id,]
test  <- mortality[-id,]

pars  <- list(X = train$Day,
              Y = train$LMR,
              Xtest = test$Day,
              Ytest = test$LMR)

myMSE <- function(lambda, pars){
    model <- loess(pars$Y ~ pars$X, enp.target = lambda)
    Ypred <- predict(model, newdata = pars$Xtest)
    MSE   <- sum((pars$Ytest - Ypred)^2) / dim(test)[1]
    k     <<- k + 1
    return(MSE)
}

lambda <- seq(0.1, 40, 0.1)
MSE <- numeric()
k   <- 0
for(i in 1:length(lambda)){
  MSE[i] <- myMSE(lambda[i],pars)
}

plot(MSE,
     pch = ifelse(MSE == MSE[which.min(MSE)], 4, 1),
     cex = ifelse(MSE == MSE[which.min(MSE)], 1.2, 0.5),
     col = ifelse(MSE == MSE[which.min(MSE)], "red", "blue"))

df <- data.frame(lambda, MSE)
list("optimal MSE" = df$MSE[which.min(MSE)],
     "optimal lambda value" = df$lambda[which.min(MSE)],
     "optimal lambda location in the sequence" = which.min(MSE),
     "Number of Iterations" = k)

k <- 0
opt1 <- optimize(myMSE, interval = c(0.1,40), tol = 0.01, pars)
list("optimal MSE" = opt1$objective,
     "optimal lambda location in the sequence" = opt1$minimum,
     "Number of Iterations" = k)

k <- 0
opt2 <- optim(par = 35, fn = myMSE, method = "BFGS", pars = pars)
list("optimal MSE" = opt2$value,
```

```
     "Number of  Iterations" = k,
     "Number of function evaluations" = unlist(opt2)[[3]],
     "Number of gradient evaluations" = unlist(opt2)[[4]])


##Question 2: Maximizing likelihood
load("C:/Users/WizzCon/Desktop/Machine Learning/Workshop/6. Computational Statistics/1. Labs/Data/data.

N     <- length(data)
mu    <- (1/N) * sum(data)
sigma <- sqrt((1/N) * sum((data-mu)^2))
paste("Mu =", mu)
paste("Sigma =", sigma)

like <- function(pars,data){
  N   <- length(data)
  mu  <- pars[1]
  sd  <- pars[2]
  log <- -(1/(sd * (sqrt(2*pi)))^N) * exp(-((1/2 * sd^2) * sum((data-mu)^2)))
  return(log)
}
print(like(pars = c(mu,sigma), data), digits = 22)

log_like <- function(pars,data){
  N   <- length(data)
  mu  <- pars[1]
  sd  <- pars[2]
  log <- -log(1/(sd * (sqrt(2*pi)))^N) + (1/2) * sum((data-mu)^2/sd^2)
  k   <<- k + 1
  return(log)
}
log_like(pars = c(mu,sigma), data)

gr <- function(pars,data){
  N  <- length(data)
  mu <- pars[1]
  sd <- pars[2]
  mu_gr <- -1 * -(sum(mu-data) / sd^2)
  sd_gr <- -1 * ((-N/sd) + sum((data-mu)^2) / sd^3)
  gr <- c(mu_gr,sd_gr)
  t  <<- t + 1
  return(gr)
}
t <- 0
gr(pars = c(mu,sigma), data)

k <- 0
t <- 0
op1 <- optim(par = c(0,1), fn = log_like, method = "CG", data = data)
k1 <- k
t1 <- t

k <- 0
```

```r
t <- 0
op2 <- optim(par = c(0,1), fn = log_like, method = "BFGS", data = data)
k2 <- k
t2 <- t

k <- 0
t <- 0
op3 <- optim(par = c(0,1), fn = log_like, gr = gr, method = "CG", data = data)
k3 <- k
t3 <- t

k <- 0
t <- 0
op4 <- optim(par = c(0,1), fn = log_like, gr = gr, method = "BFGS", data = data)
k4 <- k
t4 <- t

df    <- data.frame(unlist(op1), unlist(op2), unlist(op3), unlist(op4))
calls <- as.data.frame(matrix(c(k1, k2, k3, k4, t1, t2, t3, t4), 2, byrow = TRUE))
names(calls) <- names(df)
df    <- rbind(df,calls)

rownames(df) <- c("Mu", "Sigma", "Negative Log-likelihood", "function evaluations", "gradient evaluatio
colnames(df) <- c("CG w/o gradient", "BFGS w/o gradient", "CG with gradient", "BFGS with gradient")

knitr::kable(t(df[1:3,]))
knitr::kable(t(round(df[c(4,5,7,8),],0)))
```